Utilisation de sous-systèmes carrés et denses pour le filtrage de CSP numériques

Ignacio Araya, Gilles Trombettoni, Bertrand Neveu

Utilisation de sous-systèmes bien contraints pour le filtrage de CSP numériques

Ignacio Araya, Gilles Trombettoni, Bertrand Neveu
INRIA, Université de Nice-Sophia, CERTIS

{Ignacio.Araya,neveu,trombe}@sophia.inria.fr

Résumé

Quand les méthodes par intervalles sont appliquées aux systèmes d'équations sur les réels, deux principaux types de filtrage sont utilisés pour réduire l'espace de recherche. Quand le système est carré, l'algorithme de Newton sur intervalles se comporte comme une contrainte globale sur tout le système $n \times n$. D'autre part, les algorithmes de filtrage, provenant de la programmation par contraintes, effectuent une boucle de propagation de type AC3, où les contraintes sont traitées itérativement *une par une*.

Cet article étudie la possibilité de filtrer des *sous-systèmes bien contraints* $k \times k$, avec $1 \leq k \leq n$. Nous donnons les définitions théoriques des consistances partielles *Box-k-consistance* et *kB-consistance structurelle* qui généralisent la box-consistance. Nous montrons que les sous-systèmes bien contraints se comportent comme des contraintes globales $k \times k$ qui peuvent apporter un filtrage supplémentaire par rapport à Newton sur intervalles et aux sous-systèmes 1×1 standard. La contraction obtenue pendant le processus de résolution des sous-systèmes permet également d'apprendre des points de choix multi-dimensionnels, c.-à-d. de bissecter simultanément plusieurs domaines de variables dans l'arbre de recherche. Des expérimentations soulignent les gains en temps CPU obtenus sur des systèmes décomposés et structurés par rapport aux algorithmes connus.

1 Introduction

Quand les méthodes par intervalles sont appliquées aux systèmes d'équations sur les réels, deux principaux types de filtrage sont utilisés pour réduire l'espace de recherche. Quand le système est carré , c.-à-d., contient n variables contraintes par n équations, l'algorithme de Newton sur intervalles (`I-Newton`) se comporte comme une contrainte globale sur une linéarisation du système entier $n \times n$. Les algorithmes de filtrage, provenant de la programmation par contraintes, effectuent une boucle de propagation de type AC3, où les contraintes sont traitées itérativement une par une par une procédure *revise* qui, comme par exemple `BoxNarrow`, filtre un sous-système 1×1.

Cet article étudie la possibilité de filtrer des sous-systèmes bien contraints $k \times k$, avec $1 \leq k \leq n$. Nous définissons de nouvelles consistances partielles sur des sous-systèmes qui sont bien contraints, ce qui ajoute une restriction structurelle à la kB-consistance [13]. La propriété d'être structurellement bien contraint ainsi que les bases sur les techniques à intervalles sont définies dans la partie 2. La Box-k-consistance et la S-kB-consistance présentées dans la partie 6 s'avèrent être une généralisation de la Box-consistance [2].

Nous détaillons dans la partie 4 les procédures *revise* qui filtrent un sous-système pour le rendre Box-k ou S-kB consistant. Cette procédure développe un arbre de recherche local, à l'intérieur du sous-système et utilise un algorithme `I-Newton` local. Ces procédures *revise* ont des points communs avec l'algorithme proposé dans [7]. Leur algorithme réalise aussi une recherche arborescente où chaque nœud est filtré avant de retourner une approximation extérieure des sous-boîtes obtenues. Mais il est appliqué au système d'équations entier et non à des sous-systèmes bien contraints.

La partie 5 présente en détail comment les arbres de recherche locaux construits dans les sous-systèmes permettent à une stratégie de résolution d'apprendre des points de choix multidimensionnels dans l'arbre de recherche global, c.-à-d. de couper les domaines de plusieurs variables simultanément. Ces points de choix multidimensionnels sont appelés *multisplits*. Des expérimentations prometteuses montrent l'apport de notre approche pour des systèmes de contraintes numériques (NCSP) décomposés et structurés qui sont fréquents

en pratique. Nous montrons aussi que le schéma filtrage S-kB + multisplit généralise l'algorithme de retour arrière entre blocs (`IBB`) [17, 16] dédié à la résolution de systèmes décomposés (comme par exemple certains systèmes de contraintes géométriques).

2 Fondements

Les algorithmes présentés dans cet article ont pour but de résoudre des systèmes d'équations, ou plus généralement des CSP numériques.

Définition 1 *Un* **CSP numérique** *(NCSP)* $P = (X, C, B)$ *comprend un ensemble* X *de* n *variables et un ensemble* C *de contraintes sur ces variables. Chaque variable* $x_i \in X$ *peut prendre une valeur réelle dans l'intervalle* $[x_i]$ *et* B *est le produit cartésien (appelé* **boîte***)* $[x_1] \times ... \times [x_n]$*. Une solution de* P *est une affectation des variables de* X *satisfaisant toutes les contraintes de* C*.*

Comme les ordinateurs ne savent pas représenter les nombres réels, les bornes d'un intervalle $[x_i]$ doivent être définis comme des nombres à virgule flottante. Les opérations ensemblistes comme l'inclusion et l'intersection sont définis sur des boîtes. Un opérateur d'enveloppe `Hull` est souvent utilisé pour calculer une approximation englobante de l'union de plusieurs boîtes.

Définition 2 *Soit* $S = \{[b_1], ..., [b_n]\}$ *un ensemble de boîtes sur les mêmes variables.*
Nous appelons enveloppe de S*, notée* `Hull`*(*S*), la boîte minimale incluant* $[b_1], [b_2], ..., [b_n]$*.*

Certains NCSP admettent un nombre fini de solutions. Ce sont généralement des systèmes $n \times n$ d'équations indépendantes. Ils sont **carrés** dans le sens que n équations contraignent n variables. Le graphe de contraintes biparti correspondant, dont les sommets sont les variables d'une part et les contraintes d'autre part, vérifie la propriété structurelle suivante.

Définition 3 *Un NCSP est* **(structurellement) bien-contraint** *si son graphe biparti correspondant admet un couplage parfait [9].*

Pour trouver toutes les solutions d'un NCSP avec des techniques par intervalles, le processus de résolution commence avec une boîte initiale représentant l'espace de recherche. Il construit un arbre de recherche en bissectant la boîte courante, c.-à-d. en la coupant en deux sur une dimension (une variable), créant deux sous-boîtes. A chaque nœud de l'arbre de recherche, des algorithmes de filtrage (ou contraction) réduisent la boîte courante. Ces algorithmes comprennent des algorithmes `I-Newton` provenant de la communauté d'analyse numérique [10, 15] et des algorithmes de contraction provenant de la communauté de programmation par contraintes. Le processus s'arrête quand on a obtenu des **boîtes atomiques** de taille inférieure à la précision demandée ϵ sur chaque dimension.

Le nouvel algorithme de contraction présenté dans cet article généralise l'algorithme bien connu `Box` qui établit la propriété de *Box-consistance* [2] définie ainsi :

Définition 4 *Un NCSP* (X, C, B) *est* **box-consistant** *si chaque paire* (c, x) *est box-consistante (*$c \in C$*,* $x \in X$ *et* x *est une des variables contraintes par* c*).*

Considérons une paire (c, x)*, où* $c(x, y_1, ..., y_a) = 0$ *est une équation d'arité* $a+1$*. Soit* c' *l'équation* c *où les variables* y_i *sont remplacées par leur domaine courant dans* B *:* $c'(x) = c(x, [y_1], ..., [y_a]) = 0$*.* (c, x) *est box-consistant si :*

- $0 \in c'([\underline{[x]}, +]) = c([\underline{[x]}, +], [y_1], ..., [y_a])$ *;*
- $0 \in c'([-, \overline{[x]}]) = c([-, \overline{[x]}], [y_1], ..., [y_a])$*.*

$\underline{[x]}$*, resp.* $\overline{[x]}$*, est la borne inférieure, resp. supérieure, de* $[x]$*.* $[\underline{[x]}, +]$ *est l'intervalle étroit (d'un u.l.p.) dont les bornes sont* $\underline{[x]}$ *et le nombre flottant suivant.* $[-, \overline{[x]}]$ *est l'intervalle dont les bornes sont le nombre précédant* $\overline{[x]}$ *et* $\overline{[x]}$*.* c *représente indifféremment l'expression analytique sur les réels et la fonction étendue aux intervalles.*

En pratique, l'algorithme `Box` réalise une propagation de type AC3. Pour chaque paire (c, x), il réduit les bornes de $[x]$ de telle sorte que la nouvelle borne inférieure (resp. supérieure) soit la plus petite solution (resp. la plus grande) de l'équation à une variable $c'(x) = 0$. Les procédures *revise* existantes utilisent un principe de *rognage* pour réduire $[x]$. Des tranches $[s_i]$ dans $[x]$ sont enlevées si $c([s_i], [y_1], ..., [y_a])$ ne contient pas 0 (on utilise pour cela `I-Newton` univarié).

Une autre consistance partielle bien connue est la kB-consistance [13].

Définition 5 *Une contrainte* c *d'un NCSP* (X, C, B) *est* **2B-consistante** *(ou hull-consistante) si la boîte* B *est la plus petite boîte englobant toutes les solutions de* c*. Un NCSP* P *est* **kB-consistant** *si, pour toute variable* x*, les fermetures par* $(k-1)B$*-consistance de* $P_{\underline{[x]}}$ *(c.-à-d.,* P *avec* $[x]$ *remplacé par* $\underline{[x]}$*) et de* $P_{\overline{[x]}}$ *ne sont pas vides.*

L'algorithme `HC4` [2] calcule en gros la 2B-consistance d'un NCSP **ternarisé** dans lequel on a introduit des variables auxiliaires (et des équations additionnelles) pour faire disparaître les occurrences

multiples des variables. La 2B-consistance et la kB-consistance sont similaires à resp. l'arc-consistance et la consistance SAC [8] restreintes aux bornes des intervalles.

3 Consistances partielles Box-k et S-kB

Consistance Box-k

Comme nous l'avons expliqué auparavant, la Box-consistance produit une boîte englobante pour des sous-systèmes 1×1 (c, x). La Box-k-consistance introduite dans cet article généralise la Box-consistance en produisant une approximation extérieure de sous-systèmes $k \times k$ bien contraints. La Box-k-consistance se focalise sur des sous-systèmes suffisamment denses pour pouvoir réduire fortement l'espace de recherche.

Définition 6 *Soit $P' = (X', C', B')$ un sous-système du NCSP $P = (X, C, B)$ ($|X'| = |C'| = k$), obtenu par remplacement des* **variables d'entrée** *(les variables présentes dans au moins une contrainte de C' mais n'appartenant pas à X') par leur intervalle courant dans B. Les contraintes C' sont des équations et le sous-graphe (X', C') est connexe et structurellement* **bien contraint**, *c.-à-d., admet un couplage parfait.*

Le sous-système P' est **Box-k-consistant** *si il existe une boîte de taille 1 u.l.p. sur chaque face de la k-boîte B' telle que toute contrainte c dans C' est "satisfaite", c.-à-d., $0 \in c(X')$.*

Un NCSP est Box-k-consistant si tous ses sous-systèmes bien contraints de taille inférieure ou égale à k sont Box-k-consistants.

La figure 1-gauche montre un exemple de sous-système 2×2. La boîte extérieure est Box-consistante puisqu'elle approxime chaque contrainte de manière optimale. La boîte intérieure est Box-2-consistante puisqu'elle approxime de manière optimale l'ensemble des 6 solutions *épaisses* des deux contraintes. Les contraintes sont *épaisses* car les variables d'entrée (ex : w_1, w_2, w_3) sont remplacées par des intervalles. La figure 1-droite montre le couplage parfait (arêtes en gras) du sous-graphe correspondant.

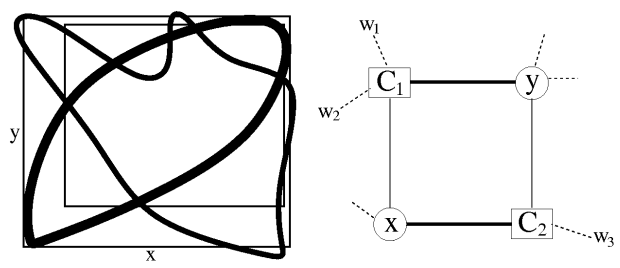

FIG. 1 – Illustration de la Box-2-consistance

Remarques

Pour des raisons d'efficacité, les consistances partielles des NCSP sont en pratique définies avec une précision ϵ, qui doit alors remplacer 1 u.l.p. dans les définitions précédentes.

Restreindre les sous-systèmes à des sous-graphes d'équations bien contraints a deux avantages. D'abord, cela permet un filtrage fort dans des sous-graphes spécifiques, ce qui est utile pour des NCSP peu denses ou pour des systèmes (globalement) sous-contraints, par exemple des systèmes mélangeant équations et inégalités. Ensuite, cela permet d'utiliser `I-Newton` pour contracter le sous-système. Comme la complexité en temps de la méthode `I-Newton` est cubique dans le nombre de variables, son application peut être trop chère pour de très grands NCSP. C'est pourquoi il est intéressant dans ce cas d'utiliser `I-Newton` uniquement pour des sous-systèmes.

Consistance S-kB

Cette nouvelle consistance partielle est une restriction de la bien connue kB-consistance, ou plus précisément $(k+2)$B-consistance, pour laquelle seuls les sous-systèmes bien contraints de taille k sont considérés. Elle est plus forte que la Box-k-consistance car non seulement les k-points sont consistants pour les contraintes à l'intérieur d'un sous-système $k \times k$ (comme Box-k), mais aussi les k-points sont 2B-consistants vis à vis du système entier.

Définition 7 *Soit $P' = (X', C', B')$ un sous-système du NCSP $P = (X, C, B)$ ($|X'| = |C'| = k$), obtenu par remplacement des* **variables d'entrée** *par leur intervalle courant dans B. Les contraintes dans C' sont des équations ; le sous-graphe (X', C') est connexe et structurellement bien-contraint.*

Le sous-système P' est **structurellement kB-consistant** *(en abrégé :* **S-kB-consistant***) si il existe une boîte $B\epsilon$ de taille 1 u.l.p. sur chaque face de la k-boîte B' telle que :*

- *toute contrainte c de C' est "satisfaite" : $0 \in c(X')$, et*
- *le système entier $(X, C, B\epsilon)$ est 2B-consistent.*

Il est bien connu que la 3B-consistance est plus forte que la Box-consistance [6], même si la 3B-consistance est calculée sur le système ternarisé, c.-à-d., si l'algorithme `HC4` est utilisé pour enlever des tranches dans le processus de rognage. Ce théorème peut être immédiatement généralisé comme suit.

Proposition 1 *La S-kB-consistance est plus forte que la Box-k-consistance. La $(k+2)B$-consistance est plus forte que la S-kB-consistance ($k \geq 1$).*

Pour comprendre pourquoi la S-kB-consistance (définition 7) est liée à la (k+2)-B-consistance, nous devons "déplier" la définition (5) récursive de cette dernière.

Apports de ces consistances structurelles partielles

L'exemple suivant montre comment une contraction obtenue sur un sous-système $k \times k$ peut être plus forte que sur des sous-systèmes 1×1 (2B-consistance) et sur le système entier $n \times n$ réalisé par `I-Newton`. Considérons le NCSP $P = (\{x, y, z\},\ \{x - y = 0,\ x + y + z = 0,\ (z-1)(z-4)(2x+y+2) = 0\},\ \{[-10^6, 10^6], [-10^6, 10^6][-10, 10]\})$.

Les contracteurs `HC4` (ou `Box`) et `I-Newton` sur P ne filtrent pas la boîte. Le calcul de la Box-2-consistance sur le sous-système 2×2 ({x,y}, {x-y=0, x+y+z=0}) réduit les intervalles de x et y à $[-5, 5]$ comme on le voit sur la figure 2. De plus, si des étapes de bissection sont utilisées pour trouver les solutions, il ne faut que deux points de choix pour isoler les 3 solutions $\{(\frac{-2}{3}, \frac{-2}{3}, \frac{4}{3}),\ (-0.5, -0.5, 1),\ (-2, -2, 4)\}$. Soulignons que `I-Newton` sur le système entier ne contracte pas la boîte car elle contient plusieurs solutions, tandis que `I-Newton` sur le sous-système 2×2 la contracte, car elle ne contient qu'une solution (épaisse en z) : le segment en gras. Bien sûr, ce petit exemple est seulement présenté à des fins didactiques. Les expérimentations décrites dans la partie 6 montrent sur de plus grandes instances non linéaires les gains de ces consistances partielles par rapport à d'autres consistances partielles comme la 3B-consistance [13].

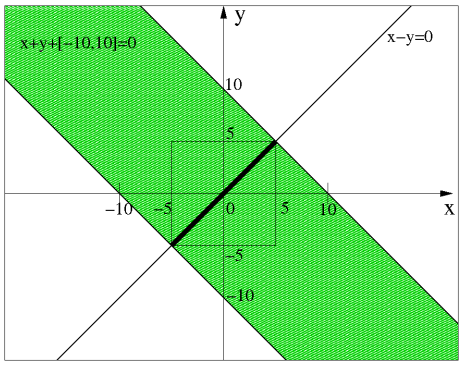

FIG. 2 – Illustration du sous-système de taille 2 avec $z = [-10, 10]$ comme variable d'entrée. {[-5,5],[-5,5]} est box-2-consistant pour le (sous-)système constitué des 2 contraintes $x - y = 0$ et $x + y + [z] = 0$.

4 Algorithmes de contraction utilisant des sous-systèmes bien contraints comme contraintes globales

Réaliser ces consistances partielles structurelles sur tous les sous-systèmes d'une taille donnée k est trop cher et contre-productif en pratique. Le nombre de sous-systèmes de taille k est grand et cet aspect exponentiel demeure même si on se limite aux systèmes bien contraints [3].

C'est pourquoi nous avons conçu une propagation de type AC3 qui gère des sous-systèmes de différentes tailles : des sous-systèmes de taille 1 mais aussi des sous systèmes bien contraints plus grands. Ces sous-systèmes sont ainsi similaires à des contraintes globales [18, 12] qui peuvent être définies par l'utilisateur ou automatiquement (voir partie 6).

Tous les sous-systèmes sont d'abord mis dans une queue de propagation et contractés en séquence. Quand le domaine d'une variable est réduit d'un ratio plus grand que ρ_{propag}, tous les sous-systèmes contenant cette variable sont ajoutés dans la queue, s'ils n'y sont pas déjà. Cette propagation est spécialisée par la procédure *revise* utilisée pour contracter les sous-systèmes de taille supérieure à 1 et détaillée ci après.

4.1 Les procédures `Box-k-revise` et `S-kB-revise`

La procédure *revise* est basée sur une méthode *branch & prune*, qui limite la bissection aux k variables de sortie X du sous-système, et utilise une stratégie de recherche en *largeur d'abord*. A la fin du parcours de cet arbre de recherche local, la boîte courante est remplacée par la boîte enveloppe des feuilles de l'arbre local. L'algorithme `Boxk-SkB-Revise` est une procédure générique qui calcule la Box-k-consistance du sous-système si le paramètre C contient les k équations du sous-système alors qu'elle effectue la S-kB-consistance du sous-système si C contient toutes les équations du NCSP. Cette procédure gère une liste L de nœuds, qui sont les feuilles de l'arbre local. Une feuille l de L a 3 attributs principaux : $l.box$ désigne l'espace de recherche (n-dimensionnel) associé au nœud ; $l.precise$ est un booléen indiquant si $l.box$ a atteint la précision ϵ sur toutes les dimensions (ϵ est aussi la précision requise pour la solution globale) ; $l.certified$ est un booléen indiquant si $l.box$ contient une solution unique. Le paramètre `box` est la boîte globale courante (espace de recherche) quand la procédure *revise* est appelée.

Un processus combinatoire (recherche arborescente) est réalisé par la boucle `while`. A chaque itération, une feuille de L, qui n'est ni *precise* ni *certified*, est choisie, bissectée et les deux nouvelles sous-boîtes sont contractées. La recherche s'arrête quand toutes les feuilles sont

Algorithm 1 `Boxk-SkB-Revise` (**in-out** L, `box`; **in** X, C, ϵ, `subContractor`, τ_{leaves}, $\tau_{\rho_{io}}$)

```
UpdateLocalTree(L, box, X, C, ε, subContractor)
L' ← {l ∈ L s.t. ¬l.certified and ¬l.precise and
ProcessLeaf(l,X,C,τ_ρio)}
while 0 < L'.size and L.size < τ_leaves do
  l ← L'.front() /* Choisit une feuille en largeur d'abord */
  (l1, l2) ←bisect(l, X)
  contract(l1, subContractor, X, C, ε)
  contract(l2, subContractor, X, C, ε)
  if l1.box ≠ ∅ then L.pushBack(l1) end if
  if l2.box ≠ ∅ then L.pushBack(l2) end if
  L.remove(l)
  L' ← {l ∈ L s.t. ¬l.certified and ¬l.precise
  and ProcessLeaf(l,X,C,τ_ρio)}
end while
box ←hull(L) /* Enveloppe de l'union de toutes les boîtes
l.box, l ∈ L */
```

étiquetées comme *certified* ou *precise* ou quand on a atteint une limite τ_{leaves} (fixée expérimentalement à 10) pour le nombre de feuilles. τ_{leaves} limite les besoins en mémoire (voir partie 4.5) et permet de propager rapidement les réductions obtenues aux autres sous-systèmes.

Une feuille est simplement choisie en largeur d'abord. Nous avions d'abord essayé une heuristique plus sophistiquée pour choisir une grosse boîte sur le bord de l'enveloppe des différentes feuilles. L'idée était de maximiser le gain en volume sur la boîte globale courante au cas où la feuille sélectionnée serait éliminée par filtrage. Nous avons écarté cette généralisation multi-dimensionnelle de l'algorithme `BoxNarrow` (qui rogne les bornes de l'intervalle traité dans l'algorithme de `Box`) car elle n'apportait pas de gains de performance significatifs.

Algorithm 2 `contract`(**in-out** l; **in** `subContractor`, X, C, ϵ)

```
if ¬l.precise then
  if ¬l.certified then
    subContractor(l.box)
  end if
  if l.box ≠ ∅ and I-Newton(l.box,X) then
    l.certified ← true
  end if
  if maxDiameter(l.box) < ε then
    l.precise ← true
  end if
end if
```

La procédure `contract` est paramétrée principalement par la procédure de contraction `subContractor` utilisée (`HC4` ou `3BCID` dans nos expérimentations). Ce sous-contracteur porte sur le système C d'équations, c'est-à-dire sur le sous-système $k \times k$ (pour `Box-k-Revise`) ou sur le système entier (pour `S-k-B-Revise`). Après un appel à `subContractor`, `I-Newton` est appelé sur le sous-système $k \times k$. Si `I-Newton` certifie qu'il existe une solution unique dans une feuille, c.-à-d. si `I-Newton` contracte *l.box* et renvoie *true*, cette feuille est étiquetée comme *certified*.

4.2 Réutilisation de l'arbre local (procédure `UpdateLocalTree`)

Une version plus simple de l'algorithme 1 n'appelait pas la procédure `UpdateLocalTree` et initialisait simplement la liste L avec la boîte courante. Cependant, plutôt que de réaliser un effort de recherche intensif dans un seul sous-système, nous avons préféré propager rapidement les réductions obtenues aux autres sous-systèmes. Ainsi, la procédure `UpdateLocalTree` réutilise l'arbre local (c.-à-d., ses feuilles) qui a été sauvé lors d'un appel précédent à l'algorithme 1. Chaque feuille dans la liste courante L est alors mise à jour en l'intersectant avec la boîte courante et en la contractant avec `subContractor`.

Algorithm 3 `UpdateLocalTree` (**in-out** L; **in** `box`, X, C, ϵ, `subContractor`)

```
if L = ∅ then
  /* Initialise la racine de l'arbre local avec la boîte courante */
  L ← {Leaf(box)}
else
  for all l ∈ L do
    /* Met à jour et contracte chaque feuille de l'arbre local */
    if l.box ≠ (l.box ∩ box) then
      l.box ← l.box ∩ box
      contract(l, subContractor, C, ε))
      if l.box = ∅ then L.remove(l) end if
    end if
  end for
end if
```

En fait, les feuilles des arbres locaux sont également sauvegardées dans l'arbre de recherche global. A cette fin, la liste L est implantée comme une structure de données *réversible* (*"backtrackable"*) mise à jour en cas de retour arrière. Cela évite de refaire plusieurs fois le même travail dans les sous-systèmes, en particulier quand l'heuristique *multisplit* est utilisée (voir partie 5).

4.3 Traitement paresseux des feuilles

Les premières expérimentations nous ont montré que le traitement d'une feuille dans un arbre local,

c'est-à-dire le fait de la bissecter et de contracter les deux sous-boîtes, était souvent contre-productif. Nous avons alors défini un ratio entrée/sortie ρ_{io} pour décider si une feuille donnée devait être traitée dans l'arbre local.

$$\rho_{io}(B, I, O, F) = \frac{Max_{x \in I}(smear(x))}{Max_{x \in O}(smear(x))}$$

La fonction `ProcessLeaf` calcule ρ_{io} sur une feuille. Si ce ratio est plus grand qu'un seuil $\tau_{\rho_{io}}$, la feuille n'est alors pas traitée par la procédure *revise* courante.

ρ_{io} est basé sur la fonction *smear* [11] définie par : $smear(x) := Max_{f \in F}(|\frac{\partial f}{\partial x}| \times Diam(x))$. Cette fonction est souvent utilisée pour choisir la prochaine variable à bissecter dans les NCSP (celle qui a la plus grande évaluation pour cette fonction *smear*).

On peut alors expliquer le dénominateur de ρ_{io} de la manière suivante : les variables de sortie avec une grande évaluation *smear* (entraînant un petit ratio ρ_{io}) indiquent une grande contraction potentielle quand elles sont bissectées dans l'arbre local au sous-système. Désirer un petit impact des variables d'entrée est moins intuitif. Nous comprenons que des domaines d'entrée larges conduisent généralement à des domaines de sortie (correspondant aux boîtes de feuilles) larges aussi et donc à une faible réduction. Cet argument est aussi valable pour les dérivées des fonctions. Pour illustrer ce point, prenons un sous-système de taille 1 comme $0.001\,y + x^2 - 1 = 0$ (x est la variable de sortie ; $[x] = [y] = [-1, 1]$), avec $\rho_{io} = \frac{0.002}{4} = 0.0005$. Après une bissection sur x, la contraction de ce sous-système produit un tout petit intervalle pour x. On obtiendrait un grand intervalle pour x si l'on considérait le sous-système $y + x^2 - 1 = 0$ avec $\rho_{io} = \frac{2}{4} = 0.5$.

4.4 Paramètres utilisateurs de la procédure *revise*

Il est apparu dans les expérimentations que le réglage de $\tau_{\rho_{io}}$ avait un impact important sur le temps de calcul, de telle sorte que ce paramètre est devenu le principal paramètre à régler pour les algorithmes de Box-k ou S-kB consistance. Les deux autres paramètres à fixer sont `subContractor` et τ_{leaves}. Les expérimentations décrites en partie 6 montrent que le sous-contracteur `3BCID` est généralement un bon choix, et τ_{leaves} a été fixé empiriquement, son impact sur le temps de calcul étant beaucoup moins important que celui de $\tau_{\rho_{io}}$.

4.5 Propriétés de la procédure *revise*

La proposition suivante formalise la correction et les complexités en mémoire et en temps de la procédure `Boxk-SkB-Revise`.

Proposition 2 *Soit $P' = (X', C', B)$ un sous-système du CSP $P = (X, C, B)$, avec $|X| = n$, $|C| = m$, $|X'| = |C'| = k$.*

La procédure `Boxk-SkB-Revise`*, appelée avec $\tau_{leaves} = +\infty$ et $\tau_{\rho_{io}} = +\infty$, rend P' Box-k-consistant si l'ensemble des équations est C' (respectivement S-kB-consistant si l'ensemble des équations est C).*

Soit Diam le plus grand diamètre des intervalles de B. Soit $d = log_2(\frac{Diam}{\epsilon})$, le nombre maximum de fois qu'un intervalle donné doit être bissecté pour atteindre la précision requise ϵ [1].

La complexité en mémoire de `Boxk-SkB-Revise` *est $O(k\,\tau_{leaves})$.*

Le nombre d'appels à `subContractor` *est $O(k\,d\,\tau_{leaves})$.*

Preuve. La correction se base sur le processus combinatoire réalisé par la procédure `Boxk-SkB-Revise`. Appelée avec le sous-système de contraintes C' et avec $\tau_{leaves} = +\infty$, la procédure calcule toutes les boîtes atomiques de précision ϵ dans le sous-système avant de retourner leur enveloppe, réalisant ainsi la consistance globale de P'.

La complexité en mémoire vient de la recherche en largeur d'abord qui doit stocker les $O(\tau_{leaves})$ feuilles de l'arbre local. La procédure *revise* traite des boîtes n-dimensionnelles, mais pour économiser de la mémoire, elle ne stocke que les k intervalles d'un sous-système $k \times k$.

Le nombre d'appels au sous-contracteur est borné par le nombre de nœuds dans l'arbre local. Le nombre de feuilles de cet arbre est τ_{leaves} (correspondant aux boîtes *vivantes* qui peuvent contenir des solutions) plus le nombre de feuilles *mortes* éliminées par filtrage et qui se ramassent à la pelle. Pour chaque feuille vivante l, le nombre de nœuds créés dans l'arbre pour atteindre l est au plus $2 \times d \times k$ car la racine doit être au plus bissectée d fois dans chacune de ses k dimensions. Bien que de nombreux nœuds internes sont partagés par plusieurs feuilles vivantes, cela borne le nombre d'appels à l'opérateur de sous-filtrage par $O(k\,d\,\tau_{leaves})$. □

5 Découpage multidimensionnel

Il est apparu que les procédures `Box-k` et `S-kB` *revise* ont non seulement un effet de contraction, mais donnent la perspective d'une nouvelle manière de faire des points de choix, c'est-à-dire de construire l'arbre de recherche global. Cette nouvelle stratégie de découpage de l'espace de recherche est appelée découpage multidimensionnel (en abrégé *multisplit*).

[1] d est généralement compris entre 20 et 60 dans les NCSP existants.

Définition 8 *Considérons un sous-système $k \times k$ P' défini dans un NCSP $P = (X, C, B)$. Considérons un ensemble S de m boîtes associé à P' tel que S contienne toutes les solutions de P et que les m boîtes obtenues par projection sur P' des boîtes de S soient deux à deux disjointes.*

Un multisplit de dimension k consiste à découper l'espace de recherche B en les m boîtes de S.

En pratique, les m boîtes correspondent aux feuilles de l'arbre local d'un sous-système. A la fin d'une propagation `S-kB`, notre stratégie de résolution fait un choix entre une bissection classique et un *multisplit*. Si tous les sous-systèmes ont un ratio ρ_m supérieur à un seuil donné τ_m, alors on effectue une bissection standard. Sinon, nous choisissons le sous-système avec le plus petit ρ_m, et nous remplaçons la boîte courante par l'ensemble L des m feuilles de l'arbre local.

$$\rho_m = \frac{\sum_{l \in L} Volume(l)}{Volume(Hull(L))}$$

La procédure *multisplit* généralise une procédure utilisée par `IBB` (cf. partie 6.1). `IBB` réalise un *multisplit* quand il a trouvé les m solutions dans un bloc donné. La différence ici est qu'un *multisplit* peut avoir lieu avec des boîtes dont la taille n'a pas atteint la précision requise.

6 Expérimentations

Les algorithmes `Box-k` et `S-kB` ont été implantés dans l'outil de résolution de contraintes sur intervalles `Ibex`, bibliothèque logicielle libre écrite en `C++` [5, 4]. La variante avec *multisplit* (`msplit`) réalise un découpage multidimensionnel du sous-système avec le ratio ρ_m minimum, pourvu que $\rho_m < \tau_m$=0.99. Tous les compétiteurs sont implantés dans la même bibliothèque, ce qui rend la comparaison équitable.

6.1 Expérimentations sur des problèmes décomposés

Dix problèmes *décomposés*, décrits dans [17, 16], apparaissent dans le tableau 1. Ils ont été auparavant décomposés par des algorithmes de graphe (*eq*) comme le couplage maximal, ou par des algorithmes géométriques utilisant la rigidité (*geo*). Ils peuvent être résolus efficacement par la méthode `IBB` [17, 16].

Brève description de `IBB`

`IBB` est dédié à des systèmes décomposés, c'est-à-dire des systèmes peu denses d'équations qui ont préalablement été décomposés en une séquence de blocs/sous-systèmes bien contraints irréductibles [1]. L'algorithme de retour arrière inter blocs (`IBB`) traite chaque bloc dans l'ordre de la séquence. Il entrelace des étapes de contraction (réalisés par `HC4` et `I-Newton`) et des bissections à l'intérieur du bloc jusqu'à ce que des boîtes atomiques (solutions du bloc) soient obtenues. Des points de choix sont alors effectués : les variables du bloc sont remplacées par une des boîtes atomiques et sont donc considérées comme constantes dans les blocs suivants.

La procédure `Box-k-Revise` plus *multisplit* représente une généralisation de `IBB` dans le sens que les domaines des variables d'entrée d'un sous-système ne sont plus nécessairement atomiques et qu'un *multisplit* n'est pas toujours réalisé après le traitement d'un sous-système. En d'autres termes, le traitement des blocs par IBB n'est pas une procédure *revise*, c'est une procédure *ad hoc* dans un algorithme dédié aux systèmes décomposés. Appliquée aux systèmes décomposés, notre nouvelle approche n'exploite pas par ailleurs l'ordre entre les blocs qui procure à IBB une heuristique de découpage utile.

Protocole expérimental

Chaque stratégie `Box-k` a été réglée avec 6 jeux de valeurs différents pour les paramètres : $\tau_{\rho_{io}}$ est 0.01, 0.2 ou 0.8 (0.01 est toujours la meilleure valeur pour les systèmes décomposés ; la précision ρ_{propag} utilisée dans la propagation `HC4` est 1% ou 10% ; Tous les autres paramètres ont été fixés empiriquement : la précision ρ_{propag} dans la propagation `Box-k` est toujours 10% ; le nombre maximum τ_{leaves} de feuilles dans un arbre local est 10 ; le nombre de tranches de `3BCID` dans `S-kB(3BCID)` est 10. Pour être équitable, les paramètres des algorithmes compétiteurs ont été réglés de sorte que 8 essais ont été réalisés pour `Box` et `HC4`, et 16 essais pour `3BCID`. Pour tous les tests, le déclenchement de I-Newton (taille du diamètre maximum à partir duquel I-Newton est lancé) est 10, et le même ordre des variables est utilisé pour la bissection dans une stratégie à tour de rôle (sauf pour `IBB` et pour `Box-k` avec *multisplit*, qui ont leurs propres heuristiques).

Les sous-systèmes gérés par la propagation sont définis automatiquement. Les blocs irréductibles produits par la décomposition de `IBB` constituent les sous-systèmes gérés par la propagation `Box-k`.

Résultats

Les stratégies basées sur `HC4`, `Box` et `3BCID` suivies par `I-Newton` ne sont pas du tout compétitives avec `Box-k` et `IBB` pour les systèmes décomposés testés. La comparaison de `Box-k`[2] avec `IBB` est très positive

[2] En nous basant sur nos précédents travaux sur `IBB`, nous avons concentré l'effort à l'intérieur des sous-systèmes, écartant la procédure `S-kB` revise pour cette catégorie de problèmes.

TAB. 1 – Résultats expérimentaux sur les problèmes IBB. Les 3 premières colonnes indiquent le nom du système, son nombre de variables et son nombre de solutions. Les 3 colonnes suivantes donnent le temps de calcul (en haut) et le nombre de boîtes (en bas), c.-à-d., le nombre de points de choix, obtenus sur un processeur Intel 6600 2.4 GHz par les stratégies existantes basées sur HC4, Box ou 3BCID suivies par I-Newton (entre deux bissections réalisées avec une heuristique à tour de rôle pour le choix de variable). Les 4 dernières colonnes donnent les résultats obtenus par nos algorithmes sur la même machine : Box-k paramétré par subContractor=HC4 ou subContractor=3BCID, avec *multisplit* (msplit) ou sans. Seule une propagation Box-k est lancée entre deux bissections.

Problème	#vars	#sols	**HC4**	**Box**	**3BCID**	**IBB**	**BoxK(HC4)**		**BoxK(3BCID)**	
								msplit		**msplit**
Chair(eq)	178	8	>3600	>3600	>3600	**0.27**	>3600	16.5*	>3600	**0.52**
1x15,1x13,1x9,5x8,3x6,...								575		15
Latham(eq)	102	96	>3600	>3600	39.9	**0.17**	**0.94**	1.35	1.5	1.08
1x13,1x10,1x4,25x2,25x1					587		839	199	991	189
Ponts(eq)	30	128	33.4	33.4	1.89	**0.59**	6.85	8.19	0.79	**0.71**
1x14,6x2,4x1			20399	20399	357		783	231	307	231
Ponts(geo)	38	128	44.1	44.1	2.6	**0.16**	2.01	0.31	1.45	**0.39**
13x2,12x1			18363	18363	685		6711	767	6711	767
Sierp3(geo)	124	198	>3600	>3600	77.5	**0.62**	49.0	**1.38**	52.5	1.77
44x2,36x1					1727		84169	1513	84169	1513
Star(eq)	46	128	>3600	>3600	4.9	**0.05**	35.6	**0.12**	44.0	0.26
3x6,3x4,8x2					283		44195	263	44023	263
Tangent(eq)	28	128	77	77	2.1	**0.08**	1.74	**0.08**	1.87	0.14
1x4,10x2,4x1			390903	390903	753		12027	255	12235	255
Tangent(geo)	42	128	–	–	7.38	**0.08**	0.80	0.19	0.80	**0.19**
2x4,11x2,12x1					859		1415	251	1407	251
Tetra(eq)	30	256	1281	1281	12.3	**0.63**	33.6	1.06	13.57	**0.76**
1x9,4x3,1x2,7x1			607389	607389	1713		4619	483	2243	483
Sierp3(eq)	cf. tableau 2					>5000	cf. tableau 2			

car les temps de calcul pour IBB sont réellement les meilleurs temps obtenus par toutes les variantes de cet algorithme. De plus, aucun *timeout* n'est atteint par Box-k+multisplit et IBB est en moyenne seulement 2 fois plus rapide que Box-k (au plus 6 sur Latham). Comme prévu, les résultats confirment que le découpage multidimensionnel est toujours pertinent pour les problèmes décomposés.

Pour le problème Sierp3(eq) (la fractale de Sierpinski au niveau 3) une décomposition équationnelle fait apparaître un gros bloc irréductible 50×50 de contraintes de distance. Cela rend IBB avec HC4 inefficace sur ce problème. Il est cependant à noter qu'une variante de IBB avec un filtrage entre blocs (IBF) [16] utilisant 3B (variante confidentielle car souvent inefficace) réussit à résoudre ce problème en 330 secondes.

6.2 Expérimentations sur des systèmes structurés

Huit systèmes *structurés* apparaissent dans le tableau 2. Ce sont des chaînes de contraintes d'arité raisonnable [14]. Ils sont appelés *structurés* car ils ne sont pas suffisamment creux pour être décomposés , c'est-à-dire que le système contient un seul bloc irréductible, rendant IBB sans objet. Une brève analyse manuelle du graphe de contraintes pour chaque problème nous a conduits à définir quelques sous-systèmes bien contraints de taille raisonnable (entre 2 et 10). De la même façon, nous avons remplacé le bloc 50×50 de Sierp3(eq) par des sous-systèmes 6×6 et 2×2 pour S-kB.

Les stratégies standard basées sur HC4 ou Box suivies par I-Newton ne sont généralement pas compétitives avec S-kB sur les problèmes testés. La comparaison avec 3BCID est bonne, le gain variant entre 0.7 et 12. Notre stratégie de résolution basée sur S-kB apparaît donc comme étant un algorithme hybride robuste, jamais très loin derrière 3BCID et quelquefois nettement meilleur. Le nombre de boîtes obtenu souligne le pouvoir de filtrage additionnel apporté par nos algorithmes traitant les sous-systèmes bien contraints. De nouveau, autoriser les multisplits semble être la meilleure option.

6.3 Apports des dispositifs sophistiqués dans les procédures *revise*

Nous montrons finalement les tableaux 3 et 4 pour évaluer les apports individuels de deux points : le paramètre $\tau_{\rho_{io}}$ utilisé par la procédure ProcessLeaf et la liste réversible des feuilles permettant de réutiliser le travail effectué dans les sous-systèmes.

Pour chaque case dans les deux tableaux, on indique le meilleur résultat en temps obtenu avec les deux sous-contracteurs HC4 et 3BCID. La première ligne de résultats correspond à la procédure Boxk-SkB-Revise im-

TAB. 2 – Résultats sur les problèmes structurés. Le même protocole a été suivi, sauf qu'une propagation `S-kB` (et non `Box-k`) donne les meilleurs résultats quand elle suit une contraction `3BCID` et I-Newton (entre deux bissections).

Problème	#vars	#sols	**HC4**	**Box**	**3BCID**	**S-kB(HC4)**		**S-kB(3BCID)**	
							msplit		**msplit**
Bratu	60	2	58	626	48.7	47.0	**33.0**	135	126
29x3			15653	13707	79	39	17	43	25
Brent	10	1015	1383	127	17.0	28.5	20.2	44.9	31.0
2x5			7285095	42191	9849	2975	4444	4585	1309
BroydenBand	20	1	>3600	0.17	**0.11**	0.45	0.15	0.91	0.31
1x6,3x5				1	21	4	19	17	3
BroydenTri	30	2	1765	**0.16**	0.25	0.22	0.24	0.39	0.29
6x5			42860473	63	25	11	19	9	3
Reactors	30	120	>3600	>3600	288	340	315	81.4	**67.5**
3x10					39253	14576	10247	1038	788
Reactors2	10	24	>3600	>3600	28.8	**9.5**	12.3	10.4	12.2
2x5					128359	4908	10850	4344	5802
Sierp3(eq)	83	6	>3600	>3600	4917	>3600	>3600	>3600	**389**
1x14,6x6,15x2,3x1					44803				218
Trigexp1	30	1	>3600	13	**0.08**	**0.08**	**0.08**	**0.08**	0.09
6x5				27	1	1	1	1	1
Trigexp2	11	0	1554	>3600	83.7	**81.2**	85.7	105	83.0
2x4,2x3			2116259		16687	15771	16755	3797	2379

TAB. 3 – Apport sur les problèmes `IBB` de la structure de données réversible (BT) et de $\tau_{\rho_{io}}$ dans la stratégie `Box-k`. $\tau_{\rho_{io}} = \infty$ signifie que les feuilles du sous-système seront toujours traitées dans la procédure *revise*.

	Chair	**Latham**	**Ponts(eq)**	**Ponts(geo)**	**Sierp3(geo)**	**Star**	**Tan(eq)**	**Tan(geo)**	**Tetra**
BT, $\tau_{\rho_{io}}$	0.52	1.08	0.71	0.31	1.38	0.12	0.08	0.19	0.76
¬ BT, $\tau_{\rho_{io}}$	10.8	4.61	1.51	1.27	23.9	2.34	0.71	1.58	2.13
BT, $\tau_{\rho_{io}} = \infty$	23.4	4.71	2.60	1.00	23.8	1.67	1.09	1.81	3.57
¬ BT, $\tau_{\rho_{io}} = \infty$	24.2	6.60	2.80	1.11	23.9	2.40	1.15	1.82	3.54

plantée ; les lignes suivantes correspondent à des versions plus simples pour lesquelles au moins un des deux dispositifs a été enlevé ou les deux (dernière ligne). Le multisplit est permis dans tous les tests.

On peut faire trois observations principales. D'abord, quand on observe un gain significatif apporté par ces dispositifs sur un système donné, alors ce système est traité efficacement par rapport aux compétiteurs (cf. tableaux 1 et 2). Ensuite, $\tau_{\rho_{io}}$ semble avoir un plus grand impact sur la performance que la liste réversible mais la différence est faible.

Enfin, plusieurs systèmes sont seulement légèrement améliorés par un des deux points tandis que le gain est significatif quand les deux sont mis ensemble. Ceci est vrai pour la plupart des problèmes `IBB`. Sur ces systèmes, entre 2 bissections dans l'arbre de recherche, il arrive souvent que le travail à l'intérieur de plusieurs sous-systèmes conduise à identifier des boîtes atomiques (d'autres sous-systèmes ne sont pas complètement explorés grâce à $\tau_{\rho_{io}}$). Bien qu'un multisplit ne soit effectué que sur un seul des sous-systèmes, le travail réalisé sur les autres est sauvegardé grâce à la liste réversible.

7 Conclusion

Nous avons proposé de nouveaux algorithmes de filtrage pour traiter des sous-systèmes bien contraints $k \times k$ dans un NCSP. Des appels à un `I-Newton` $k \times k$ et des bissections à l'intérieur de tels sous-systèmes sont utiles pour mieux contracter les NCSP décomposés et structurés. De plus, les arbres de recherche locaux, à l'intérieur des sous-systèmes, permettent à la stratégie de recherche globale d'apprendre des points de choix intéressants consistant à couper plusieurs domaines simultanément.

Les stratégies de recherche basées sur les propagations `S-kB` ou `Box-k` et le découpage multi-dimensionnel ont trois paramètres principaux : le choix entre `Box-k` et `S-kB` (`Box-k` semble meilleur uniquement pour les systèmes décomposés), le choix du sous-contracteur (`3BCID` semble être souvent un bon choix), et $\tau\rho_{io}$. Ce dernier paramètre apparaît comme le plus important à régler.

Les premières expérimentations sur des systèmes décomposés et structurés ont montré que nos nouvelles stratégies de résolution étaient plus efficaces que les stratégies standard basées sur `HC4`, `Box` ou `3BCID` (avec `I-Newton`) pour ces problèmes. `Box-k`+multisplit peut

TAB. 4 – Apport sur les problèmes structurés de la structure de données réversible (BT) et de $\tau_{\rho_{io}}$ (dans la stratégie `S-kB`).

		Bratu	**Brent**	**BroyB.**	**BroyT.**	**Reac.**	**Reac.2**	**Sierp3B(eq)**	**Trigexp1**	**Trigexp2**
BT,	$\tau_{\rho_{io}}$	33.0	20.2	0.15	0.24	67.5	12.2	389	0.08	83.0
¬ BT,	$\tau_{\rho_{io}}$	33.2	21.0	0.14	0.23	97.7	12.0	411	0.07	85.0
BT,	$\tau_{\rho_{io}} = \infty$	33.9	23.8	0.38	0.28	164	13.1	519	0.1	103
¬ BT,	$\tau_{\rho_{io}} = \infty$	33	28.7	0.40	0.38	401	18.7	533	0.07	148

être vu comme une généralisation de `IBB`. Il peut aussi résoudre des systèmes décomposés de grande taille avec des blocs assez petits en moins d'une seconde, et peut par ailleurs traiter des systèmes structurés que `IBB` ne peut pas résoudre.

Les sous-systèmes ont été ajoutés automatiquement pour les NCSP décomposés, mais manuellement pour les NCSP structurés. Dans cet article, nous avons validé le fait que le traitement de sous-systèmes pouvait apporter une contraction supplémentaire et des points de choix multi-dimensionnels pertinents. La prochaine étape est de sélectionner automatiquement un ensemble de sous-systèmes. Nous pensons que des algorithmes de graphe prenant en compte un critère similaire à ρ_{io} peuvent être adaptés pour conduire à des heuristiques efficaces.

Références

[1] S. Ait-Aoudia, R. Jegou, and D. Michelucci. Reduction of Constraint Systems. In *Compugraphic*, 1993.

[2] F Benhamou, Goualard F., L. Granvilliers, and J.-F. Puget. Revising Hull and Box Consistency. In *Proc. of ICLP*, pages 230–244, 1999.

[3] C. Bliek, B. Neveu, and G. Trombettoni. Using Graph Decomp. for Solving Continuous CSPs. In *Proc. CP, LNCS 1520*, pages 102–116, 1998.

[4] G. Chabert. `www.ibex-lib.org`, 2009.

[5] G. Chabert and L. Jaulin. Contractor Programming. *Artificial Intelligence*, Available online 18 March 2009.

[6] H. Collavizza, F. Delobel, and M. Rueher. Comparing Partial Consistencies. *Reliable Computing*, 5(3) :213–228, 1999.

[7] J. Cruz and P. Barahona. Global Hull Consistency with Local Search for Continuous Constraint Solving. In *Proc. EPIA, LNAI 2258*, pages 349–362, 2001.

[8] R. Debruyne and C. Bessière. Some Practicable Filtering Techniques for the Constraint Satisfaction Problem. In *Proc. IJCAI*, pages 412–417, 1997.

[9] A.L. Dulmage and N.S. Mendelsohn. Covering of Bipartite Graphs. *Canadian Journal of Mathematics*, 10 :517–534, 1958.

[10] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis*. Springer, 2001.

[11] R.B. Kearfott and M. Novoa III. INTBIS, a portable interval Newton/Bisection package. *ACM Trans. on Mathematical Software*, 16(2) :152–157, 1990.

[12] Y. Lebbah, C. Michel, M. Rueher, D. Daney, and J.P. Merlet. Efficient and safe global constraints for handling numerical constraint systems. *SIAM Journal on Numerical Analysis*, 42(5) :2076–2097, 2005.

[13] O. Lhomme. Consistency Tech. for Numeric CSPs. In *IJCAI*, pages 232–238, 1993.

[14] J-P. Merlet. `www-sop.inria.fr/coprin/ logiciels/ALIAS/Benches/benches.html`, 2009.

[15] A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge Univ. Press, 1990.

[16] B. Neveu, G. Chabert, and G. Trombettoni. When Interval Analysis helps Interblock Backtracking. In *Proc. CP, LNCS 4204*, pages 390–405, 2006.

[17] B. Neveu, C. Jermann, and G. Trombettoni. Inter-Block Backtracking : Exploiting the Structure in Continuous CSPs. In *Selected papers WS COCOS, LNCS 3478*, pages 15–30, 2005.

[18] J.-C. Régin. A Filtering Algorithm for Constraints of Difference in CSPs. In *Proc. AAAI'94*, pages 362–367, 1994.